IN THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently amended) A method for loop optimization within a dynamic compiler system, comprising:

discovering each index expression within a loop portion;

determining which arrays are accessed using the index expressions ~~discovered by the discovering~~;

for each of the arrays accessed using the index expressions, sorting the index expressions by ~~the~~ trip counter and offset portions of the index expressions; and

creating a loop structure ~~wherein~~ using iteration splitting wherein a plurality of loops are generated, each loop of the plurality of loops being based on an original loop structure of the loop portion, and wherein ~~is reduced to eliminate~~ at least one of an upper or lower range check is eliminated in at least one loop of the loop structure, the loop structure being determined based on the sorted index expressions;

wherein the creating a loop structure comprises creating a pre-loop structure based on ~~an~~ the original loop structure of the loop portion, wherein the pre-loop structure is capable of testing indexing expressions for underflow; generating a main loop structure having indexing expressions based on the original loop structure, wherein the indexing expressions cannot produce an underflow, and wherein the indexing expressions cannot produce an overflow; and creating a post-loop structure based on the original loop structure, wherein the post-loop structure is capable of testing indexing expressions for overflow.

2. (original) A method as recited in claim 1, wherein the pre-loop structure includes an array boundary test.

3. (original) A method as recited in claim 2, wherein the post-loop structure includes an array boundary test.

4. (original) A method as recited in claim 3, wherein the main loop structure does not include an array boundary test.

5. (original) A method as recited in claim 1, further including the operation of compiling a computer program during execution of the computer program.

6. (original) A method as recited in claim 5, further including the operation of interpreting lines of the computer program during execution of the computer program.

7-26 (canceled)

27. (Currently amended) A method for range check elimination for a loop portion of a computer program, the method comprising:

discovering each index expression within the loop portion;

determining which arrays are accessed using the index expressions ~~discovered by the discovering~~;

for each of the arrays accessed using the index expressions, sorting the index expressions by the trip counter and offset; and

creating a loop structure having a plurality of loops wherein iteration splitting is ~~reduced~~ used to eliminate at least one of an upper or lower range check in at least one loop of the loop structure, the loop structure being determined based on the sorted index expressions.

28. (Previously presented) The method of claim 27 wherein the program is a compiler's internal representation of bytecode.

29. (Previously presented) The method of claim 28 further comprising transforming the computer program into native executable code.

30 (Previously presented) The method of claim 27 wherein the loop structure comprises a pre-loop based on the loop portion wherein indexing expressions are boundary tested for underflow only, a main loop based on the loop portion wherein indexing expressions are not boundary tested, and a post-loop based on the loop portion wherein indexing expressions are boundary tested for overflow only.

31. (Currently amended) A method for loop optimization in a computer program, the method comprising:

analyzing the program to discover loops; and

for each of the loops, performing loop cleanup and loop transformations, wherein loop cleanup comprises moving loop invariant operations outside the loop body and wherein loop transformations comprises:

discovering each index expression within the loop;

determining which arrays are accessed using the index expressions discovered by the discovering;

for each of the arrays accessed using the index expressions, sorting the index expressions by ~~the~~ trip counter and offset portions of the index expressions; and

creating a loop structure wherein iteration splitting is ~~reduced~~ used to eliminate at least one of an upper or lower range check in at least one loop of the loop structure, the loop structure being determined based on the sorted index expressions.

32. (Previously presented) The method of claim 31 wherein the program is a compiler's internal representation of bytecode.

33. (Previously presented) The method of claim 32 further comprising transforming the computer program into native executable code.

34 (Previously presented) The method of claim 31 wherein the loop structure comprises a pre-loop based on the loop portion wherein indexing expressions are boundary tested for underflow only, a main loop based on the loop portion wherein indexing expressions are not boundary tested, and a post-loop based on the loop portion wherein indexing expressions are boundary tested for overflow only.

35. (Currently amended) A method for executing a bytecode program, the method comprising:

optimizing a loop structure of the subroutine by performing loop transformations, the optimizing comprising discovering each index expression within the loop, determining which arrays are accessed using the index expressions ~~discovered by the discovering~~, for each of the arrays accessed using the index expressions, sorting the index expressions by ~~the~~ trip counter and offset, and creating a loop structure wherein iteration splitting is ~~reduced~~ <u>used</u> to eliminate at least one of an upper or lower range check in at least one loop of the loop structure, the loop structure being determined based on the sorted index expressions.

36. (Previously presented) The method of claim 35 further comprising:

receiving a bytecode in an interpreter;

determining whether native code corresponding to the bytecode is available;

when the native code is not available, incrementing a bytecode counter;

interpreting the bytecode when the bytecode counter is below a threshold;

compiling a subroutine containing the bytecode into native code when the bytecode counter is above the threshold, the compiling comprising the optimizing; and

executing the native code after the compiling or when the native code is determined to be available.

37. (Previously presented) The method of claim 36 wherein the compiling further comprises generating an internal representation of the subroutine prior to the optimizing.

38. (Previously presented) The method of claim 35 wherein the loop structure comprises a pre-loop based on the loop portion wherein indexing expressions are boundary tested for underflow only, a main loop based on the loop portion wherein indexing expressions are not boundary tested, and a post-loop based on the loop portion wherein indexing expressions are boundary tested for overflow only.